Published in final edited form as:
J Comput Chem. 2016 May 5; 37(12): 1048–1058. doi:10.1002/jcc.24290.

Fast search algorithms for Computational Protein Design

Seydou Traoré[1,2,3], **Kyle E. Roberts**[4], **David Allouche**[5], **Bruce R. Donald**[4], **Isabelle André**[1,2,3], **Thomas Schiex**[5], and **Sophie Barbe**[1,2,3,*]

[1]Université de Toulouse; INSA, UPS, INP; LISBP, 135 Avenue de Rangueil, F-31077 Toulouse, France

[2]INRA, UMR792, Ingénierie des Systèmes Biologiques et des Procédés, F-31400 Toulouse, France

[3]CNRS, UMR5504, F-31400 Toulouse, France

[4]Department of Biochemistry, Department of Computer Science, Department of Chemistry, Duke University, Durham, NC, USA

[5]Unité de Mathématiques et Informatique Appliquées de Toulouse, UR 875, INRA, F-31320 Castanet Tolosan, France

Abstract

One of the main challenges in Computational Protein Design (CPD) is the huge size of the protein sequence and conformational space that has to be computationally explored. Recently, we showed that state-of-the-art combinatorial optimization technologies based on Cost Function Network (CFN) processing allow to speed up provable rigid backbone protein design methods by several orders of magnitudes. Building up on this, we improved and injected CFN technology into the well-established CPD package *Osprey* to allow all *Osprey* CPD algorithms to benefit from associated speedups. Because *Osprey* fundamentally relies on the ability of A^* to produce conformations in increasing order of energy, we defined new A^* strategies combining CFN lower bounds, with new Side Chain Positioning (SCP)–based branching scheme. Beyond the speedups obtained in the new A^*-CFN combination, this new branching scheme enables a much faster enumeration of sub-optimal sequences, far beyond what is reachable without it. Together with the immediate and important speedups provided by CFN technology, these developments directly benefit to all the algorithms that previously relied on the DEE/A^* combination inside *Osprey* and make it possible to solve larger CPD problems with provable algorithms.

Introduction

Computational Protein Design (CPD) has become a valuable tool for creating proteins with desired biophysical and functional properties and for assessing our understanding of protein sequence-structure-function relationships. By combining physico-chemical models governing relations between protein amino-acid composition and the protein three-

[*]Correspondence to: Sophie Barbe, Laboratoire d'Ingénierie des Systèmes Biologiques et des Procédés – INSA; CNRS UMR5504; UMR INRA 792; 135, Avenue de Rangueil; F-31077 Toulouse cedex 4, France. Tel: +33 561 559 963; Fax: +33 561 559 400; sophie.barbe@insa-toulouse.fr.

dimensional structure with advanced computational algorithms, CPD seeks to identify one or a set of amino-acid sequences that fold into a given 3D-scaffold and that will bestow the re-designed protein with targeted properties. Such rational protein design approaches have been successfully applied to alter intrinsic properties (stability, binding affinity…) of existing proteins or to endow them with new functionalities, leading to the generation of novel enzymatic catalysts, therapeutic proteins, protein-protein interfaces and self-assembling protein structures [1]–[6]. The applications of this technology is broad, ranging from medicine, biotechnology, and synthetic biology to nanotechnologies [7].

Despite notable results, substantial methodological advances are still needed to improve CPD performances and extend its effective application. The success of CPD predictions depends on several elements, which include the biologically meaningful modelling of the design problem, the accuracy of the energy and objective functions used to assess fitness of the predicted sequence-structures, and the efficiency of the search algorithms to find solutions in a timely manner. However, to face the exponential size of the search space defined by the composition of protein sequences and conformations, CPD approaches have to strike a compromise between speed and accuracy. Most of the CPD methods rely on: 1) a coarse-graining of the structure as a sequence of discrete side chain rotamers, 2) an assumption of modest backbone conformational flexibility, where often a fixed backbone or a set of possible backbones are used, and 3) an approximation of the energy model as a pairwise decomposition. Since the problem of searching for an optimal solution (GMEC : Global Minimum-Energy Conformation) over the conformational space of rotamers and possibly backbones is NP-hard [8], a variety of methods, both meta-heuristics (Monte Carlo simulated annealing [9], [10], genetic algorithms, [11]) and provable algorithms (Dead-End Elimination (DEE), Branch-and-Bound algorithms (BB), Integer Linear Programming (ILP), Dynamic Programming (DP), [11]–[16]) have been proposed over the years. However, there is still a need for more efficient optimization techniques, capable of exploring vaster combinatorial spaces, representing more realistic and flexible protein models.

This paper focuses on exact optimization and enumeration techniques. As provable methods know when a global optimum is reached, the search can be stopped with confidence and an exact solution obtained, sometimes in significantly less time than with meta-heuristics. It has also been observed that the accuracy of meta-heuristic approaches tends to degrade in unpredictable ways as the problem size increases [9]. Finally, exact methods are the only methods which offer a provable basis for improving biophysical models. Indeed, they ensure that discrepancies between CPD predictions and experimental results come exclusively from modelling inadequacies and not from the algorithm. These properties are crucial to rationally tune the energy function, eg. on the basis of a fixed-composition full redesign [17]. Currently, the most usual provable and deterministic methods for CPD rely on the Dead-End Elimination theorem and the A^* algorithm [12]. DEE is used as a pre-processing method. It removes rotamers which are energetically dominated by other rotamers and therefore useless to identify a global optimum. This idea has later been extended to prune pairs or higher order combinations of rotamers at different residues in order to improve the pruning power [18]–[20]. However, because CPD is NP-hard, the polynomial time DEE algorithms usually cannot identify a unique sequence-conformation model. DEE preprocessing is therefore followed by an A^* search (a provable Best-First search strategy) which expands a sequence-conformation tree by tentatively assigning rotamers to residues. A^* is originally a path planning algorithm, and it has a worst-case exponential time and memory consumption. This means that it can easily choke on problems with many undominated rotamers. However it has the capacity to not only identify the GMEC but to also produce an arbitrary long energy-sorted list of solutions. Thus the same DEE/A^* can be used to find the GMEC and to produce an energy-sorted stream of sequence-conformations.

In a recent work, we have shown that the already highly challenging usual description of the CPD problem, based on rigid backbone and discrete rotamers, could be formulated and efficiently solved as a Cost Function Network (CFN) [21]–[23]. CFN algorithms are able to handle complex CPD combinatorial spaces which are out of reach of a broad range of combinatorial optimization technologies including the usual DEE/A^*, 0/1 Linear and Quadratic Programming, 0/1 Quadratic Optimization, Weighted Partial MaxSAT and Graphical Model optimization methods [21]–[23]. The *toulbar2* CFN solver provides speedups of several orders of magnitude both to provably find the GMEC and to exhaustively enumerate unsorted ensembles of near-optimal solutions (within a threshold of the GMEC), offering an attractive alternative method for CPD.

The outperformance of CFN methods opens new avenues to integrate further molecular flexibility in CPD which inevitably leads to a tremendous expansion of the conformational search space to be considered. CPD methods capable to tackle different levels of molecular flexibility have raised a growing interest [24], [25] as this has been identified as being one major cause of design failures. Indeed, the lack of molecular flexibility may lead to a significant loss of sequence space accessible to properly folded and functional proteins thus introducing some biases in sequence selection. However, in spite of its crucial importance, incorporation of more realistic (macro)molecular flexibility into CPD remains a major challenge.

Herein, we integrated the CFN technology in the well-established *Osprey* software (modified version of *Osprey* 2.0) [26], [27]. Beyond traditional energy functions, *Osprey* includes models and provable algorithms that capture macromolecular flexibility and enable reliable biophysical modelling, making it a target of choice to dissiminate CFN technology inside the CPD community. It is noteworthy that this CPD-dedicated software has been prospectively used, with experimental validation, to redesign enzymes toward non-cognate substrates [1], [28], [29], design new drugs [30], predict drug resistance mutations [2], design peptide inhibitors of protein-protein interactions [31], and design epitope-specific antibody probes [32]. All provable methods inside *Osprey* intimately depend on the capacity of A^* to produce an energy-sorted stream of solutions, something that the *toulbar2* solver does not directly provide. We therefore injected the CFN bounding technology inside A^* so that all *Osprey* algorithms, including those with flexible modelling and affinity estimation, directly benefit from the enhanced bounding, leading to better efficiency.

To further improve this new CFN-based method [21]–[23], we integrated other essential components of CFN technology, such as ordering heuristics and branching schemes. Inspired by the specific nature of the CPD problem, we further developped a new branching scheme that provides further speedups. This new branching scheme can avoid the expensive enumeration of all conformations for a given sequence, allowing to directly produce an ensemble of near-optimal sequences. Thanks to this, exhaustive libraries of near-optimal sequences and energy-sorted lists of sequences can be directly produced with a much higher efficiency.

The performances of these methods have been assessed on the design of more stable proteins and cofactor-bound proteins, as well as protein-ligand and protein-protein interfaces. The results obtained for both the GMEC identification and the enumeration of near-optimal sequence-conformations were compared to those obtained using the DEE/A^* approach implemented in *Osprey*[1] in terms of speedup of the combinatorial optimization and enumeration step.

To make this paper more self-contained, we provide hereafter a short description of the CPD underlying concepts and methods.

Background

The CPD problem formulation

The rigid backbone and discrete rotamer CPD problem is defined by: *i*) a fixed backbone of a protein structure; *ii*) a set of amino-acid residues to be designed, called ‘designable residues’; *iii*) a group of allowed amino-acids for each designable residue and their respective set of discrete low energy side chain conformations, called rotamers and *iv*) pairwise atomic energy functions to evaluate the model. Rotamers correspond to cluster centers of well represented amino-acid side chain conformations mined from a database of 3D protein structures. In the case of protein-ligand systems, the conformational flexibility of peptide ligands is also described by discrete rotamer libraries. In the case of non-peptide ligands, the treatment of organic molecule flexibility is often left to the user to pre-calculate an ensemble of low energy conformers for the ligand (used as a rotamer library) [4].

A sequence-conformation model is defined by the choice of one specific amino-acid with one associated conformation (rotamer) for each designable residue. Its total energy (E_{total}) is defined by:

$$\mathrm{E_{total}}=\mathrm{E_c}+\sum_{\mathrm{i}}\mathrm{E}(\mathrm{i_r})+\sum_{i}\sum_{j<i}\mathrm{E}(\mathrm{i_r},\mathrm{j_s}) \quad (1)$$

where E_c is a constant energy contribution capturing interactions between fixed parts of the model, $E(i_r)$ depends on rotamer r at position i (and its reference energy) and $E(i_r, j_s)$ is the pairwise interaction energy between rotamer r at position i and rotamer s at position j.

The combinatorial optimization problem is to find, in the space of all complete rotamer assignments (each defined by a sequence and a conformation), one assignment that provably minimizes E_{total}.

[1]The modified *Osprey* source code consistent with this work is available at http://www.cs.duke.edu/donaldlab/osprey.versions.php#CFNosprey

Modelling CPD as a Cost Function Network

The problem of finding the set of rotamers that will minimize the total energy (E_{total}) can be easily formulated as a Cost Function Network problem (CFN) [21]–[23].

A CFN is defined by a set of variables which are connected by a set of local cost functions [33]. Formally, a CFN is a quadruple (X, D, C, k) where $X = \{x_1, x_2, . . , x_n\}$ is a set of n variables indexed by $I = \{1, \ldots n\}$. Each variable $x_i \in X$ has a discrete domain $d_i \in D$ that defines the set of values that it can take. A set of local cost functions C defines a network over X. Each cost function $c_S \in C$ is defined over a subset of variables indexed by $S \subseteq I$ (called its scope), has a domain $\Pi_{x_i \in S}\, d_i$ and takes integer values in $\{0,1,2, . . , k\}$. The cost k represents a maximum (intolerable) cost. It can be infinite or set to a finite upper bound. Values or combinations of values that are forbidden by a cost function are simply mapped to k. The global cost of a complete assignment A is defined as the sum of all cost functions on this assignment (or k if this sum is larger than k). The problem of finding an assignment of all variables that minimizes this global cost is called the Weighted Constraint Satisfaction Problem defined by the CFN. It is usually assumed that C contains one constant cost function, with an empty scope, denoted as $c_\varnothing$. Since all cost functions in a CFN are non-negative, this constant cost function $c_\varnothing \in C$ defines a lower bound on the cost of an optimal solution. When cost functions involve at most two variables, the CFN is said to be binary and the Cost Function Network defines a graph where variables are vertices and binary scopes are edges.

As an example, we describe a simple CFN defined by a set of three variables $X = \{x_1, x_2, x_3\}$ each with the same boolean domain $d_1 = d_2 = d_3 = \{0,1\}$ and a set of four cost functions $C = \{c_\varnothing, c_{\{1,2\}}, c_{\{1,3\}}, c_{\{2,3\}}\}$. The cost functions are defined as $c_\varnothing = 1$, $c_{\{1,2\}} = |x_1 - x_2|$, $c_{\{1,3\}} = x_1 x_3$ while $c_{\{2,3\}}$ is defined by a specific cost matrix. The upper bound $k = 9$. The graph of this CFN is represented in Figure 1, together with the cost matrix defining $c_{\{2,3\}}$. This CFN, denoted as P, defines a joint cost function $P(x_1, x_2, x_3) = \min\,(c_\varnothing + c_{\{1\}}(x_1) + c_{\{1,2\}}(x_1, x_2) + c_{\{1,3\}}(x_1, x_3) + c_{\{2,3\}}(x_2, x_3), k)$. On the triple of values $x_1 = 0$, $x_2 = 2$, $x_3 = 0$, this function has cost 1 + 1 + 0 + 3 = 5.

The CPD optimization problem, in its pairwise-decomposed form, can be easily formulated as a binary CFN. Every designable amino-acid residue i is represented by a variable x_i and the set of rotamers available to the residue defines its domain d_i. Then, each energy term in E_{total} is represented as a cost function [21]–[23]. The constant term E_c is captured as the constant cost function with empty scope ($c_\varnothing$) and terms $E(i_r)$ and $E(i_r, j_s)$ are represented by unary and binary cost functions $c_{\{i\}}$ and $c_{\{i,j\}}$ respctively, each involving the variables of the corresponding designable positions. These cost functions are defined cost matrices defined from pre-computed energy matrices. Floating point energy terms can be mapped to positive integers through shifting and scaling according to desired precision [21]–[23]. Such operations preserve the set of optimal solutions and an optimal solution of the CFN is an assignment that defines a GMEC for the CPD problem. In this model, it becomes possible to interchangeably use the notions of CFN variable and CPD designable position, the notions of value of a CFN variable and CPD rotamer, the notions of complete CFN variable assignment and of CPD sequence-conformation as well as the notions of CFN cost and CPD energy.

Solving the optimization and enumeration of CPD problems

Existing approaches to provable CPD rely on the combination of several techniques. In the most usual CPD approach, DEE is first used to prune rotamers and if needed, an A^* algorithm is used to find the GMEC or enumerate sequence-conformations. The A^* algorithm is a famous path-planning search algorithm which belongs to the larger family of Branch and Bound algorithms. The idea of Branch & Bound is to split the space of all complete assignments in disjoint subspaces in some way (branching). Then a specific lower bounding mechanism is used to quickly compute an underestimation of the energy of the best assignment in each subspace. By comparing this lower bound to the energy of any complete assignment (defining an upper bound on the optimum), it is possible to prove that some subspaces do not need to be explored, because the best assignment they contain has a worst cost than the available upper bound. This way, it is possible to avoid the exhaustive exploration of all subspaces.

Because of the recursive nature of spliting spaces in subspaces through branching, all the possible subspaces can be organized in a rooted tree where the root node is the space of all possible sequence-conformations and the children of a node which is not reduced to a singleton can be obtained by applying the branching scheme. In the usual A^* algorithm used in CPD, the branching scheme is a standard *n*-ary branching. A yet unfixed designable position x_i is *chosen*. Then, for each possible rotamer at this position in the considered node, one child is created where the position x_i is set to this rotamer. Notice that this branching scheme alone does not define a unique tree as the branching scheme must choose a designable position each time a subspace is split. In CPD, it is usual to choose the same variable at each level of the tree (a static variable ordering), in their protein sequence order. A leaf node defines a complete assignment of amino-acid rotamers. An internal node defines a partial protein conformation assignment. In Figure 2, we show the structure of such a *n*-ary branching tree for a very simple CPD problem (see caption), using an arbitrary dynamic variable ordering. But other more powerful branching schemes exist. Algorithms that aim at producing a provable GMEC or enumerating complete near-optimal assignments must traverse the defined tree in some order. A simple strategy is to use *Depth-First Search* and always explore the deepest node, leading to a *Depth-First Branch and Bound* (DFBB) algorithm. Because there are typically many nodes at the same depth, a secondary criteria, called *the value ordering heuristics* can be used to choose the next explored node. The lower bound of an explored node is immediately computed. If this lower bound is larger than or equal to the cost of the best known solution (the current upper bound), the branch below this node can be safely pruned and search proceeds in the rest of the tree. Thanks to the very ordered exploration of the tree defined by depth first search, remembering the current position of the search requires only polynomial space. Ultimately, the order in which the search will proceed in the tree, whose shape is defined by the branching scheme and the variable ordering heuristics, is conditionned only by the value ordering heuristics, the lower bounding mechanism and the initial value of the upper bound *k*. Indeed it may be easy to produce an initial non naïve complete assignment that will provide a hopefully tight initial upper bound *k*, leading to more abundant pruning. Then, when only the GMEC is sought, each time a new complete assignment is explored (a sequence-conformation), its energy is used to tighten the upper bound k and improve pruning. In this case, the last encountered leaf will be optimal. If instead one whishes to enumerate all solutions within a Δ threshold of the GMEC, a two step process is followed: first the GMEC, with energy e^*, is sought using the above algorithm. Then, the upper bound k is set to $(e^* + \Delta)$ and the search restarted but without updating k when a solution is found. This is guaranteed to enumerate all sequence-conformations within Δ of the GMEC.

In CPD, A^* Best-First Search is used instead. As the name indicates, the next node to explore is always the known unexplored node with the best lower bound. This usually uniquely defines the node and no secondary value ordering criteria is needed. Because this requires to remember which nodes have been explored, Best-First search requires exponential space in the worst-case: upon exploration of a node, new children nodes are pushed in a priority list of "open" nodes (originally containing only the root) from which the smallest lower bound node will be successively extracted until a leaf is extracted. In this case, it is known that all other open nodes cannot lead to a better solution than this one: the first extracted leaf is the GMEC. Without any change in the algorithm, if the extraction of best nodes continues, the stream of leaves explored will be an energy-sorted list of sequence–conformations. The order of the exploration of the tree, whose shape is defined by the branching scheme and the variable ordering heuristics, depends on the lower bounding mechanism used. DEE/A^* relies on a simple bound [34] which we denote in the rest of the paper as the "vanilla" bound. In A^*, the lower bound is usually denoted as "the admissible heuritics".

Cost Function Network processing

In contrast with the most usual exact CPD method, where dominance analysis through the DEE theorem is widely used, the fundamental idea in CFNs relies on so-called *Local Consistencies*. From a given CFN, it is always possible to define a family of small (local) subproblems. A Local Consistency property requires that each subproblem must be sufficiently explicit about local optimal costs. As an example, the node consistency of a variable x_i with associated cost function c_i requires that d_i contains at least one value v such that $c_i(v) = 0$ and no value w such that that $c_\varnothing + c_i(w) \geq k$ (the forbidden cost). Equivalently, this means that there is at least one value that does not locally increase cost and no value that is locally infeasible. If a problem does not satisfy a given Local Consistency property, it is possible to transform it in an equivalent problem (defining the same joint cost/energy distribution) satisfying the property thanks to *Equivalence Preserving Transformations* (EPTs). An EPT is a local transformation of the CFN which can shift cost (or energy) between cost functions of intersecting scopes without changing the global energy distribution. These EPTs are iteratively applied in so-called Local Consistency enforcing algorithms that iterate EPTs until the CFN satisfies the Local Consistency property. For example, if a variable violates node consistency then by deleting infeasible values and by shifting costs to $c_\varnothing$ (*i.e.*, by subtracting the minimum of all $c_i(v)$ from each $c_i(v)$ and adding it to $c_\varnothing$), the variable can be made node consistent. Beyond node consistency, many other Local Consistency properties and associated polynomial time enforcing algorithms have been defined [35]–[37]. Depending on the locality of the property, which may apply to one variable, one cost function or more, they are called Node, Arc or higher order consistencies.

In this paper, we only consider Node and Arc consistencies. The most desirable effects of enforcing Local Consistencies is that infeasible values, which can only lead to solutions of cost larger than k, may be pruned and that the constant cost function $c_\varnothing$, which remains a lower bound on the optimum, may be increased. The amount of pruning and the strength of the lower bound may increase as the upper bound k decreases. For a given upper bound, these effects are obtained without changing the global energy distribution. Compared to Local Consistency enforcing, DEE, which has also been studied in CFN under the name of substitutability ([23], [38], [39]) does not preserve the global energy distribution as it may remove feasible sub-optimal solutions. In CFN, the pruning and non-naïve lower bounding based on $c_\varnothing$ is done incrementally at each visited node during DFBB, using various branching schemes, variable and value ordering schemes as well as specialized upper bounding algorithms. Because the amount of pruning depends on the value of the upper bound k, it is important to have a reasonable upper bound at the beginning of the tree search. This can be obtained by doing bounded tree search using Limited Discrepancy Search [40].

Our recent studies [21]–[23] highlighted the power of these CFN-based method to efficiently solve various CPD problems and hence, spurred new developments to tackle more complex and challenging CPD problems. Novel CFN-based methodological advances are provided herein. In particular, a variant of the A^* search algorithm which uses the CFN lower bound as its admissible heuristics instead of the usual CPD lower bound [41] is defined. It is named “A^*-CFN”. The performances of this new search method were compared to those of the A^* search commonly used in CPD, denoted as “A^*-vanilla”. In addition, we enhanced CPD algorithms by injecting more CFN technology such as sophisticated variable ordering heuristics (for both $A^*$and DFBB), value ordering heuristics, upper bounding and branching schemes. The specific nature of the CPD problems was specifically used to design a new family of branching strategies based on amino-acid types and Local Consistency enforcement. The open source CPD framework *Osprey* 2.0 [27] in which the CFN-based methods have been implemented, allows for discrete and continuous modelling of the protein conformation at the side-chain and backbone level [25], [42]. All those models (including the DEEPer flexibility models) enabling the consideration of more flexibility will benefit from CFN-based developments as they are ultimately based on the optimization of a pairwise energy matrix.

New CFN-based algorithms for CPD

The strategies offered in the CFN-based CPD framework developed in this work are described in Figure 3. They are defined through a customisable tree search algorithm that can rely on different branching strategies (n-ary, dichotomic, binary and a new SCP-based branching scheme), search strategies (Best First A^* and DFBB) and variable/value orderings. This algorithm always relies on Local Consistency combined with a simple DEE pruning [23] to prune rotamers and compute a lower bound at preprocessing but also at each node of the tree search. Through the use of different combinations of branching schemes and rules for updating k in DFBB, it is possible to offer a larger variety of services than the usual DEE/A^*combination.

Tree search: strategies, heuristics and branching schemes

Tree search based exact optimization and enumeration of near-optimal solutions can be achieved by different combinations of basic ingredients. In the rest of the paper, we examine specific combinations for CPD, some of which include new ingredients. There are however some crucial features that will remain unchanged in all the considered combinations. First of all, we will always use Local Consistency at preprocessing and at each node of the search. By increasingly simplifying the problem and strengthening the lower bound $c_\varnothing$, Local Consistency provides information that can be used to prune and heuristically guide the search in a tree defined by branching schemes and variable and value ordering. Because the pruning strength of Local Consistencies depends on the tightness of the upper bound k, we also always perform an initial upper-bounding using Limited Discrepancy Search [40].

Best-First Search with a CFN lower bound: the A*-CFN—The completeness of the A^* search method (A^*-vanilla) used in the DEE/A^* approach [41] relies on the use of a so-called “admissible heuristic function”: this function must provide an optimistic estimate of the energy of all complete conformations below the current node *i.e.*, must be a lower bound on the optimum of the current problem. Thus, the lower bound defined by $c_\varnothing$ and Local Consistency enforcing can be directly plugged in the A^* algorithm and replace the existing heuristics.

The lower bound used in current DEE/A^* implementations is equivalent to the CFN ‘Directed Arc Consistency Counts’ bound [43]. This lower bound has been obsoleted by several Arc Consistency properties [44]. Hence, we expect that replacing A^* lower bound with stronger Local Consistency based bounds will improve the pruning efficiency of A^*. We use Existential Directed Arc Consistency (EDAC) [44], an Arc Consistency property with a fast incremental enforcing algorithm. When Local Consistency is enforced on an open node, domains may get pruned and the updated lower bound $c\varnothing$ is used as a priority. Since upper bounding is always done as preprocessing, any developped node with a lower bound larger than the current upper bound k needs not be inserted in the queue, which saves space. This defines the A^*-CFN algorithm. The systematic enforcement of Local Consisency at each node contrasts with DEE/A^* where DEE is only used as a preprocessing pruning technique. Hence, A^*-CFN has a permanent opportunity to reduce its search space and uses a stronger lower bound than the A^* lower bound [44].

Variable and value ordering heuristics—Besides a search strategy, tree search methods also need to choose the next variable that will be used to branch. This is achieved using a variable ordering heuristics. Our algorithm supports dynamic variable ordering where a choice of variable is done at each node (and not just at each level of the tree). Variable ordering heuristics may have a tremendous effect on the efficiency of the search algorithm. They are all based on the ‘fail-first’ principle [45] : ‘To succeed, try first where you are most likely to fail’. Several measures have been used to try to evaluate the likelihood of failing by fixing a variable. One simple measure is the current size of the domain (*dom*) [45]. Under this measure, the variable which has the smallest domain should be assigned next. To take into account the number of cost functions that involve a variable (the so-called degree of the variable), more sophisticated heuristics select the variable that has the minimum ratio of the domain size over the current degree (*dom/ddeg* 46]), over the degree weighted by the number of failures observed in the past for each cost function (*dom/wdeg* 21], [47]) or by the sum of the median cost of cost functions involving the variable (*dom/cmed* [22], [23]). Additionally, the last conflict heuristics [48] simply try to select the last variable that led to inconsistency during search (if any).

Once a variable is chosen, Depth-First search and different branching scheme need to choose the next value to consider. The effect of value ordering heuristics is often less dramatic than for variables. However, a good value ordering heuristics may help to quickly find a good solution (upper bound). The most usual value ordering heuristics in CFNs is to choose first a value a that has a unary cost $c_i(a) = 0$. Such a value always exists thanks to EDAC [44].

Branching schemes—We already described n-ary branching, where branching consists in choosing a variable x_i and all its available values to fix the variable value. Another branching scheme consists in selecting a variable x_i and a value/rotamer a and branch on the fact that the variable x_i either takes the value a, or not (and it must take one of its remaining values). This is called binary branching. By exploiting results in proof theory, binary branching has been previously shown to be more powerful than n-ary branching (with a given Local Consistency being maintained at each node, binary branching may explore an exponentially smaller number of nodes than n-ary branching, and the converse is impossible [49]. n-ary branching can however be polynomially better on some problems). In Figure 4, a search tree of the same toy CPD problem is shown using binary branching.

An even more general branching method is dichotomic branching where the domain of a chosen variable i is split in two chosen sets. The split can be done in the middle of the domain range or in the middle of the unary energy range (similarly to [50]) where low energy rotamers are part of the first set and high energy rotamers form the second set.

- A new SCP-based branching scheme

Compared to CFN, where all values are considered as uniform objects, the rotamers available at a designable position can be classified according to the chemical nature of the amino-acid they define. In CPD, ultimately, it is the nature of the amino-acid that matters more than its precise conformation (as far as the optimal energy is known). Using the amino-acid nature of rotamers, it is possible to define a new branching scheme where a variable is chosen first among variables which have more than one possible amino-acid type in their available rotamers. In this case, a dichotomic split will be done between all rotamers corresponding to a *chosen* amino-acid and all other rotamers. If all variables already have rotamers with a single amino-acid identity (the protein sequence is fixed in the current node), then a simple binary branching strategy is followed. The n-ary branching counterpart of this search mechanism is achieved by simply creating a new SCP-subproblem for each amino acid and branch onto them using an amino-acid selection heuristic. As for binary branching, once all the identities are fixed, the SCP problem is solved by n-ary branching.

We propose two heuristics to select the amino-acid type (AA) that will be used to split the domain in SCP branching. The idea behind these heuristics is to try to select an amino-acid that leads to a very good solution quickly, thus allowing to tighten the upper bound k.

Inspired by the value ordering heuristics, the first heuristics selects the amino-acid associated to a rotamer v with a cost $c_i(\mathrm{v}) = 0$ (*Zero-Cost−AA*) which always exists because of EDAC enforcing. The second one selects the wild-type amino-acid if it is still available after Local Consistency enforcing (*Wt−Zero-Cost−AA*) and otherwise uses the *Zero-Cost −AA* choice. This branching incrementally reduces CPD to SCP: each time a branching is done, the domain of a variable is restricted to a single amino-acid in the first branch. Once there is no mutable position left in a given node, either a binary or *n*-ary branching scheme is applied to select rotamers. This branching strategy is denoted as SCP-based branching in what follows.

Interestingly, thanks to the separation that SCP-branching creates between a top tree that sets sequences and a large number of subtrees that solve SCP problems, it becomes possible to address the problem of the enumeration of all unique sequences within an energy threshold of the GMEC together with their optimal conformation and associated energy. As the ultimate goal of CPD is to provide a set of unique sequences to experimentalists, the restriction of the enumeration to a single conformation per sequence makes sense [22] and should allow to drastically reduce the running time because of the exponential reduction in the number of enumerated solutions. In GMEC search, the upper bound k is updated each time a new solution is found. For enumeration, the upper bound is never updated. Since SCP-based branching defines the amino acid identity at all positions prior to conformation search, it becomes possible to update the upper bound locally only (in the current SCP subproblem) in order to just identify the best conformation for the current sequence, leading to additional pruning capabilities during enumeration. This defines two new search strategies: within a DFBB search strategy, the upper bound is locally updated once there is no mutable residue left in a given SCP branch. Similarly, for A^*-CFN with SCP-branching scheme, each time there is no mutable residue left, we simply use a DFBB search to get the best solution for the current sequence before adding the node to the list of open nodes (with its optimal energy). This hybrid A^*/DFBB strategy enables A^*-CFN to automatically enumerate unique sequences in an energy sorted order (the SCP part being explored using depth first search).

Benchmark set for method performance assessment

The benchmark set of 30 CPD instances (including structural models and associated precomputed energy matrices using *Osprey*) previously prepared [22] (Table-S1) was used to assess the performances of the CFN-based methods in terms of runtime to provably identify the GMEC or enumerate sub-optimal solutions for each protein design case. This benchmark set includes a variety of protein structures, alone or in complex with a protein or a ligand, derived from high resolution structures deposited in the Protein Data Bank (PDB) [51]. These benchmarks vary by the number of designable residues (mutable or flexible) ranging from 23 to 97, leading to a wide range of sequence-conformation search spaces going from $4.36\ 10^{26}$ to $2.17\ 10^{94}$. The penultimate rotamer library was used [52]. All pairwise energy terms were pre-computed and stored using *Osprey* 2.0 [27]. The used energy functions consist in the sum of the Amber electrostatic terms (with a distance-dependent dielectric constant), van der Waals and the EEF1 implicit solvation energy term

[53]. No cutoff was used for non-bonded interactions. A more detailed description of the benchmark set can be found in ref. [22].

All computations were performed on one core of an AMD OpteronTM Processor 6176@2.3 GHz. We used 128GB of RAM and a 9,000 sec timeout. CPU times reported hereafter correspond to runtime from the pre-processing to the identification of the optimal or sub-optimal solutions. Problems not solved within the time limit were considered as using 9,000 seconds (a lower bound on the real time they would require). Unless specified otherwise, the EDAC Arc Consistency with incremental DEE (DEE^1 option) pruning [21]–[23] was activated in all CFN-based experiments (both during preprocessing and during search), the median cost variable ordering heuristic (dom/cmed) was used in all the experiments (with the "last conflict" heuristics [48]) as well as the zero unary cost value ordering heuristic and initial upperbounding by Limited Discrepancy Search [40]. The statistic metrics reported hereafter ignore the few instances for which the difference in the time needed to solve the targeted problem was essentially similar (difference of less than 1 sec).

Results and discussion

Effects of the CFN lower bound

Performances of the new A^*-CFN method (using the CFN lower bound) were assessed against those of the A^*-vanilla algorithm (usually used in CPD after a DEE pre-processing) for the provable GMEC identification problem. In order to ensure that the difference of performances between both methods is only due to the quality of the lower bound, a static variable ordering was used. The usual *n*-ary branching was used and no upper-bounding performed to stick to usual DEE/A^*conditions (favoring this combination). The criteria used for comparison are the runtime, the number of visited nodes and the number of solved problems. To fairly compare the search methods, we performed a full DEE preprocessing (*Osprey algOption=3*) and a Local Consistency preprocessing step on each instance and then applied either A^*-vanilla or A^*-CFN. Thence, both strategies start with an identical search space for each of the 30 CPD instances.

A^*-vanilla and A^*-CFN solved, respectively, 22 and 23 CPD instances before timeout (Table 1). A^*-CFN outperforms A^*-vanilla in terms of runtime for 15 out of 18 discriminating cases (with more than 1 sec. difference). If we assign an optimistic 9,000 sec to unsolved cases, A^*-CFN allowed to save 978.9 sec in average. Except for 1MJC problem which was directly solved at the end of the preprocessing step, A^*-CFN expanded fewer nodes than A^*-vanilla in all cases solved by both methods. For example, A^*-vanilla expanded 63,974 nodes to solve 1C9O case while A^*-CFN expanded only 20 nodes (corresponding to 3,199 fold decrease in the number of expanded nodes). As expected, the inexpensive A^*-vanilla lower bound led to a higher number of nodes expanded per minute than the A^*-CFN lower bound, which is however not enough to compensate for the much higher number of nodes that A^*-vanilla needs to explore for solving the CPD problem.

We then compared the performances of the DFBB we implemented in *Osprey* against those of A^*-CFN (Figure 6 and Supplementary Table S-2), both using the same lower bounding mechanism. Since both DFBB and A^*-CFN allow for it, default value and variable ordering heuristics were activated along with an initial upper bounding using Limited Discrepancy Search. *n*-ary branching was used.

Out of the 30 design cases, A^*-CFN and DFBB managed to find 24 GMEC within the timeout. DFBB was faster than A^*-CFN in 15 cases and saved in average 323.42 sec. It was also faster than A^*-vanilla in 16 cases, saving in average 1,031.43 sec. Therefore, DFBB search with incremental Local Consistency enforcing algorithm and ordering heuristics remains overall more efficient than traditional A^* search methods.

While A^*, as a Best-First search method, is known to always explore less nodes than Depth-First search when the same tree is explored with the same lower bound, DFBB can quickly improve its upper bound k and therefore pruning whereas A^* cannot. This probably explains the performance gain of the Depth-First search strategy combined with the CFN lower bound. The polynomial space use of DFBB is also likely to help in the context of CPU processors using multiple levels of increasingly slow caches.

Branching schemes

Binary, *n*-ary, dichotomic and SCP-based branching strategies were assessed against each other (Figure 7). To select the amino-acid in SCP-based branching, the *Zero-Cost−AA* amino-acid heuristic was used. Supplementary Table S-3 reports results for all 30 CPD instances.

As expected, binary branching was faster than *n*-ary branching (by an average of 84.35 sec.) and solved more CPD problems (11 over 19 discriminating problems). Compared to the (unary cost-based) dichotomic branching, it also showed a favorable average runtime gain (400.63 sec). However, the number of problems solved faster does not increase (only 5 cases out of 18). In contrast, the SCP-based branching scheme is clearly better than all other branching schemes. Out of 20 cases and all other contenders, a minimum of 18/20 cases were solved more quickly by SCP-based branching with a minimum average runtime gain of 328.25 sec per instance. Hence, these results show that SCP-based branching outperforms all the other branching schemes, both in terms of runtime and number of problems solved faster. The poor performance of dichotomic branching is somehow unexpected, since it splits apart rotamers at a given position into two groups depending on their energies.

Enumeration of near-optimal sequence-conformation ensemble

The problem of enumerating sub-optimal sequence-conformation models within a given energy range above the GMEC (arbitrarily set here to $2\ kcal.\ mol^{-1}$) was considered for all 30 CPD instances. First, we applied the A^*-CFN and DFBB schemes using the binary branching scheme and with the SCP-based branching activated (default ordering heuristics and upper bounding activated in all cases).

For the A^*-based strategies, the average runtime is 5959.4 sec and 4590.8 sec for, respectively, A^* and A^*-SCP (Figure 8). For the DFBB-based methods, the average runtime is 4432.3 sec and 3958.4 sec for, respectively DFBB and DFBB-SCP. As expected, SCP brings significant improvement to both A^*-CFN and DFBB search strategies also in the context of enumeration. It is important to remind that A^* based algorithms preserve their natural advantage of being able to produce an energy-sorted stream of sequence-conformations. Instead DFBB based algorithms produce an unsorted list that cannot be extended unless the algorithm is rerun.

Enumeration of near-optimal unique sequences

Finally, the performances of the CFN-based algorithms were also compared to enumerate ensembles of sub-optimal unique sequences using SCP-branching to separate sequence and conformation search. A comparison of runtime performances of all methods is shown in Figure 8 and more detailed results are included in Supplementary Table S-4; S-5 and S-6. Ensembles of unique sequences were generated using both A^*-SCP and DFBB-SCP methods. More cases were solved and a significant gain in runtime was always obtained when unique sequence SCP is activated, leading to 1496.7 sec average runtime for A^*-SCP-unique and 2021.2 sec for DFBB-SCP-unique (Table S-6). Consequently, SCP-based branching schemes enable enumerating unique sequences over a much larger energy window above the GMEC. Indeed, taking as an illustration the 1MJC instance and an interval of $2\ kcal.\ mol^{-1}$, the A^*-CFN required 232.6 sec to enumerate 2,110,737 sequence-conformations (4,408,541 nodes). DFBB, A^*-SCP and DFBB-SCP required, respectively, 140.7 sec (4,537,935 nodes), 143.7 sec (4,415,342 nodes) and 178.7 sec (4,555,451 nodes) to handle the same instance. However, this ensemble corresponds to only 91 unique sequences. For a larger energy interval of $20\ kcal.\ mol^{-1}$, using DFBB-SCP method restricted to the enumeration of unique sequences, 1,034 sequences were generated in 295.85 sec (with a number of nodes lower than 50,489). Because of the sheer number of conformations per sequence, this energy interval is out of reach when all conformations have to be enumerated. SCP-branching is thus an effective solution to handle CPD problems and gives access to the enumeration of unique sequences within much larger energy windows than what can be achieved using standard branching schemes.

Conclusion

This work reports the introduction of Cost Function Network optimization techniques to solve Computational Protein Design problems. Compared to prior work [21]–[23], it introduces novel search strategies in order to speedup search methods, which have been implemented in the CPD-dedicated software *Osprey*. The presented search algorithms have shown to be more efficient than optimization methods based on the DEE/A^* framework for both GMEC search and sub-optimal solutions enumeration (which is often performed in order to account for inaccuracies and approximations made in the modelling of the design problem and what is of interest to guide construction of experimentally manageable sizes of mutant libraries). In addition, the new SCP-based branching scheme makes it possible to generate unique sequences within a larger energy threshold than what was possible with other methods up to now. The implementation in *Osprey* of existing CFN algorithm together with these new methods enable novel opportunities. Indeed, these algorithms can be used in combination with all the other functionalities in this software, in particular for molecular flexibility modelling. Any macromolecular flexibility method that can be represented as an optimization problem with a single matrix can be performed with the presented CFN-based methods, even if they intrisically rely on an energy-sorted stream of solutions.

Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

Acknowledgements

This work has been funded by the "Agence Nationale de la Recherche", references ANR 10-BLA-0214 and ANR-12-MONU-0015-03. We thank the Computing Center of Region Midi-Pyrénées (CALMIP, Toulouse, France) and the GenoToul Bioinformatics Platform of INRA-Toulouse for providing computing resources and support. S. Traoré was supported by a grant from the INRA and the Region Midi-Pyrénées. K.E. Roberts and B.R. Donald supported by NIH Grant 2-R01-GM-78031-5. We thank Pablo Gainza for discussion on search algorithms in *Osprey*.

References

1. Chen C-Y, Georgiev I, Anderson AC, Donald BR. Proc. Natl. Acad. Sci. U. S. A. 2009; 106:3764–3769. [PubMed: 19228942]
2. Frey KM, Georgiev I, Donald BR, Anderson AC. Proc. Natl. Acad. Sci. U. S. A. 2010; 107:13707–13712. [PubMed: 20643959]
3. King NP, Bale JB, Sheffler W, McNamara DE, Gonen S, Gonen T, Yeates TO, Baker D. Nature. 2014; 510:103–108. [PubMed: 24870237]
4. Miklos AE, Kluwe C, Der BS, Pai S, Sircar A, Hughes RA, Berrondo M, Xu J, Codrea V, Buckley PE, Calm AM, Welsh HS, Warner CR, Zacharko MA, Carney JP, Gray JJ, Georgiou G, Kuhlman B, Ellington AD. Chem. Biol. 2012; 19:449–455. [PubMed: 22520751]
5. Karanicolas J, Kuhlman B. Computational design of affinity and specificity at protein-protein interfaces. Current Opinion in Structural Biology. 2009; 19:458–463. [PubMed: 19646858]
6. Siegel JB, Zanghellini A, Lovick HM, Kiss G, Lambert AR, St Clair JL, Gallaher JL, Hilvert D, Gelb MH, Stoddard BL, Houk KN, Michael FE, Baker D. Science. 2010; 329:309–313. [PubMed: 20647463]
7. Palomo JM, Filice M. Biotechnol. Adv. 2015; 33:605–613. [PubMed: 25560932]
8. Pierce NA, Winfree E. Protein Eng. 2002; 15:779–782. [PubMed: 12468711]
9. Voigt CA, Gordon DB, Mayo SL. J. Mol. Biol. 2000; 299:789–803. [PubMed: 10835284]
10. Kuhlman B, Baker D. Proc. Natl. Acad. Sci. U. S. A. 2000; 97:10383–10388. [PubMed: 10984534]
11. Wernisch L, Hery S, Wodak SJ. J. Mol. Biol. 2000; 301:713–736. [PubMed: 10966779]
12. Desmet J, De Maeyer M, Hazes B, Lasters I. Nature. 1992; 356:539–542. [PubMed: 21488406]
13. Gordon DB, Mayo SL. Structure. 1999; 7:1089–1098. [PubMed: 10508778]
14. Althaus E, Kohlbacher O, Lenhof H-P, Müller P. J. Comput. Biol. 2002; 9:597–612. [PubMed: 12323095]
15. Leaver-Fay A, Kuhlman B, Snoeyink J. Pac. Symp. Biocomput. 2005:16–27. [PubMed: 15759610]
16. Kingsford CL, Chazelle B, Singh M. Bioinformatics. 2005; 21:1028–1036. [PubMed: 15546935]
17. Alvizo O, Mayo SL. Proc. Natl. Acad. Sci. U. S. A. 2008; 105:12242–12247. [PubMed: 18708527]
18. Goldstein RF. Biophys. J. 1994; 66:1335–1340. [PubMed: 8061189]
19. Pierce NA, Spriet JA, Desmet J, Mayo SL. J. Comput. Chem. 2000; 21:999–1009.
20. Gordon DB, Hom GK, Mayo SL, a Pierce N. J. Comput. Chem. 2003; 24:232–243. [PubMed: 12497602]
21. Allouche D, Traoré S, André I, de Givry S, Katsirelos G, Barbe S, Schiex T. Proc. of CP-12. 2012
22. Traoré S, Allouche D, André I, De Givry S, Katsirelos G, Schiex T, Barbe S. Bioinformatics. 2013; 29:2129–2136. [PubMed: 23842814]
23. Allouche D, André I, Barbe S, Davies J, de Givry S, Katsirelos G, O'Sullivan B, Prestwich S, Schiex T, Traoré S. Artif. Intell. 2014; 212:59–79.
24. Jackson EL, Ollikainen N, Covert AW, Kortemme T, Wilke CO. PeerJ. 2013; 1:e211. [PubMed: 24255821]
25. Gainza P, Roberts KE, Donald BR. PLoS Comput. Biol. 2012; 8:e1002335. [PubMed: 22279426]

26. Donald BR. MIT Press. 2011
27. Gainza P, Roberts KE, Georgiev I, Lilien RH, Keedy Da, Chen CY, Reza F, Anderson AC, Richardson DC, Richardson JS, Donald BR. Methods Enzymol. 2013; 523:87–107. [PubMed: 23422427]
28. Lilien RH, Stevens BW, Anderson AC, Donald BR. J. Comput. Biol. 2005; 12:740–761. [PubMed: 16108714]
29. Stevens BW, Lilien RH, Georgiev I, Donald BR, Anderson AC. Biochemistry. 2006; 45:15495–15504. [PubMed: 17176071]
30. Gorczynski MJ, Grembecka J, Zhou Y, Kong Y, Roudaia L, Douvas MG, Newman M, Bielnicka I, Baber G, Corpora T, Shi J, Sridharan M, Lilien R, Donald BR, Speck NA, Brown ML, Bushweller JH. Chem. Biol. 2007; 14:1186–1197. [PubMed: 17961830]
31. Roberts KE, Donald BR, Boisguerin P, Cushing PR, Madden DR, Boisguerin P, Madden DR, Donald BR. PLoS Comput. Biol. 2012; 8:e1002477. [PubMed: 22532795]
32. Georgiev I, Acharya P, Schmidt S, Li Y, Wycuff D, Ofek G, Doria-Rose N, Luongo T, Yang Y, Zhou T, Donald B, Mascola J, Kwong P. Design of epitope-specific probes for sera analysis and antibody isolation. Retrovirology. 2012; 9:P50.
33. Schiex T, Fargier H, Verfaillie G. Int. Jt. Conf. Artif. Intell. 1995; 14:631–639.
34. Georgiev I, Lilien RH, Donald BR. J. Comput. Chem. 2008; 29:1527–1542. [PubMed: 18293294]
35. Cooper M, Schiex T. Artif. Intell. 2004; 154:199–227.
36. Larrosa J, Schiex T. Artif. Intell. 2004; 159:1–26.
37. Cooper M, de Givry S, Schiex T. 8th Int. CP-06 Work. Prefer. Soft Constraints. 2006:14p.
38. Lecoutre C, Roussel O, Dehani DE. Princ. Pract. Constraint Program. 2012:406–421.
39. de Givry S, Prestwich S, O'Sullivan B. Princ. Pract. Constraint Program. 2013; 2013
40. Harvey WD, Ginsberg ML. 14th Int. Jt. Conf. Artif. Intell. IJCAI95. 1995; 1:607–613.
41. Leach AR, Lemon AP. Proteins. 1998; 33:227–239. [PubMed: 9779790]
42. Hallen MA, Keedy DA, Donald BR. Proteins. 2013; 81:18–39. [PubMed: 22821798]
43. Wallace, R. Selected papers from the ECAI-94 Workshop on Constraint Processing. Springer; Berlin: 1995. Directed Arc Consistency Preprocessing; p. 121-137.
44. De Givry S, Zytnicki M, Heras F, Larrosa J. IJCAI. 2005; 19:84.
45. Haralick RM, Elliot GL. Artif. Intell. 1980; 14:263–313.
46. Bessière C, Régin J-C. Proc. Second Int. Conf. Princ. Pract. Constraint Program. 1996:61–75.
47. Lecoutre C, Sais L, Tabary S, Vidal V. ECAI 2006 17th Eur. Conf. Artif. Intell. August 29-September 1, 2006, Riva del Garda, Italy Incl. Prestig. Appl. Intell. Syst. (PAIS 2006) Proc. 2006:133.
48. Lecoutre C, Sa"is L, Tabary S, Vidal V. Artif. Intell. 2009; 173:1592, 1614.
49. Mitchell D. Princ. Pract. Constraint Program. 2003; 2003
50. Hong E, Lippow SM, Tidor B, Lozano-pérez T. 2009
51. Bernstein FC, Koetzle TF, Williams GJ, Meyer EF, Brice MD, Rodgers JR, Kennard O, Shimanouchi T, Tasumi M. Eur. J. Biochem. 1977; 80:319–324. [PubMed: 923582]
52. Lovell SC, Word JM, Richardson JS, Richardson DC. Proteins. 2000; 40:389–408. [PubMed: 10861930]
53. Lazaridis T, Karplus M. Proteins. 1999; 35:133–152. [PubMed: 10223287]

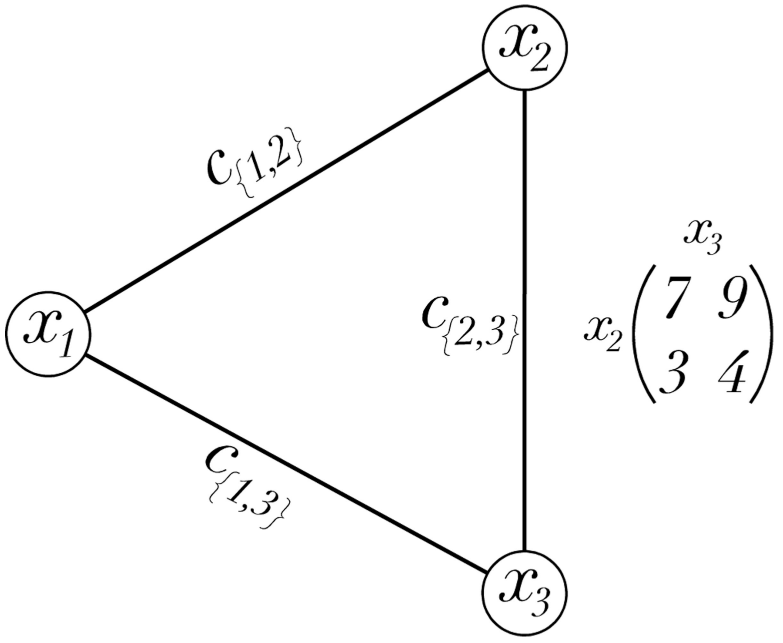
x_1
x_2
x_3
$c_{\{1,2\}}$
$c_{\{2,3\}}$
$c_{\{1,3\}}$
$\begin{array}{cc} & x_3 \\ x_2 & \begin{pmatrix} 7 & 9 \\ 3 & 4 \end{pmatrix} \end{array}$

Figure 1.
The graph of our CFN example together with the cost matrix of the cost function $\boldsymbol{c}_{\{2,3\}}$.

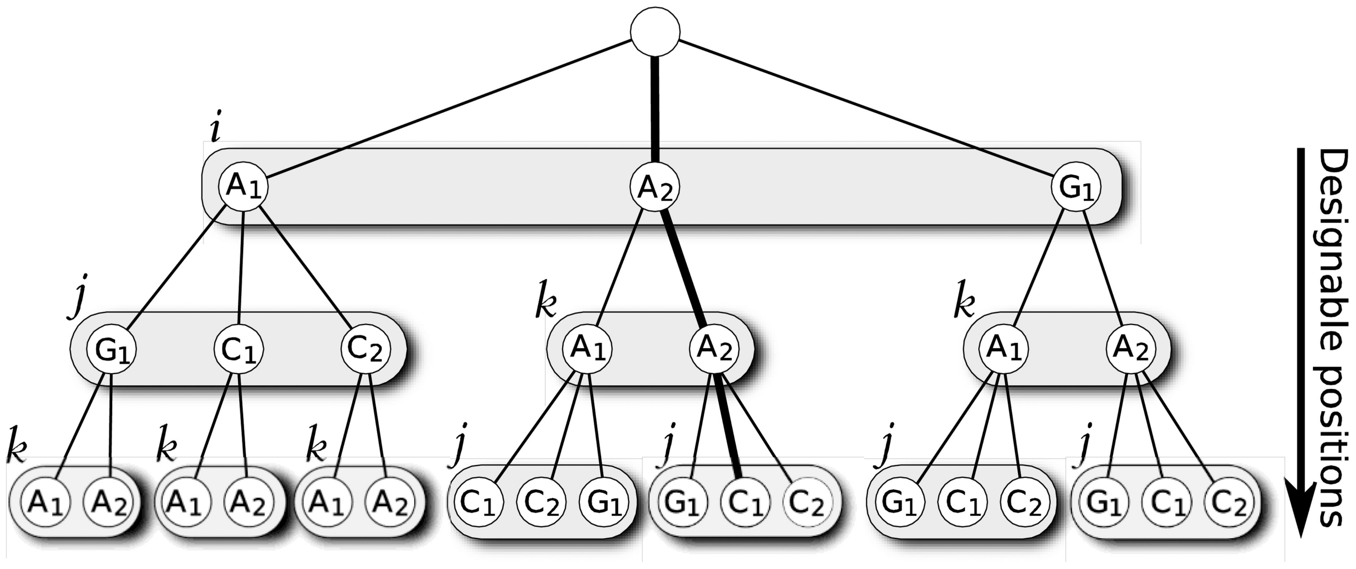

Figure 2.

Tree Search using *n*-ary branching. We assume a trivial CPD problem with three designable positions denoted ***i***, ***j***, and ***k*** with respectively 3 ($\boldsymbol{A_1}$, $\boldsymbol{A_2}$, $\boldsymbol{G_1}$), 3 ($\boldsymbol{G_1}$, $\boldsymbol{C_1}$, $\boldsymbol{C_2}$) and 2 ($\boldsymbol{A_1}$, $\boldsymbol{A_2}$) rotamers. To expand a node in children nodes (branching), a yet unfixed designable position is chosen (indicated in the top left of each grey box). The children nodes (inside the grey box) correspond each to a subspace where the position is fixed to an available rotamer. A path in the tree, as shown in bold, corresponds to a complete rotamer assignment. Note that with a dynamic variable ordering, the same designable position needs not be chosen at a given level.

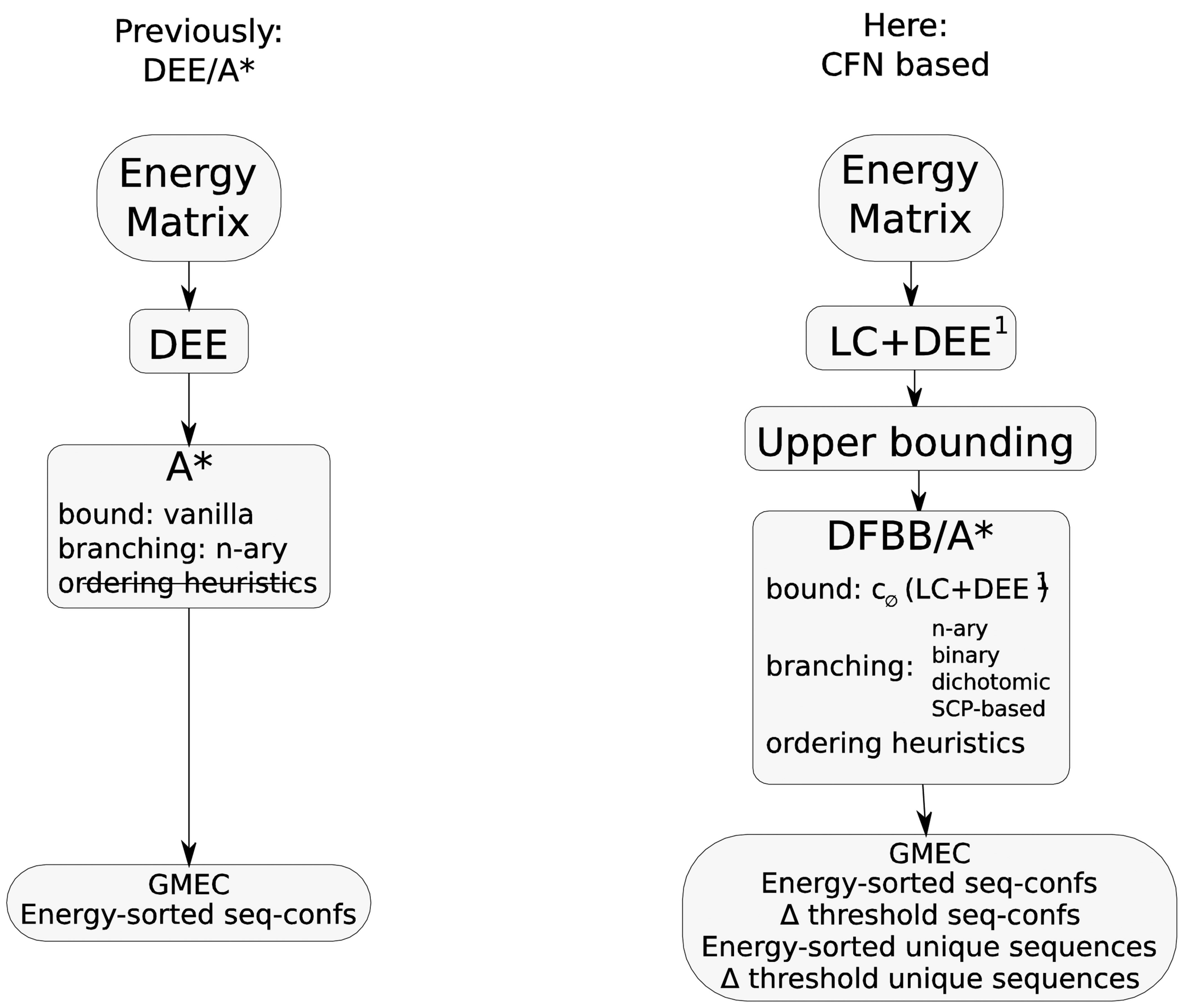

Figure 3.

CPD frameworks. By upgrading components of the usual DEE/A* combination (left) using Local Consistency (LC), new branching schemes and ordering heuristics, it is possible to increase efficiency for provably finding the GMEC and enumerating suboptimal sequence-conformations within a Δ threshold of the GMEC. It also becomes possible to directly enumerate unique near-optimal sequences (right).

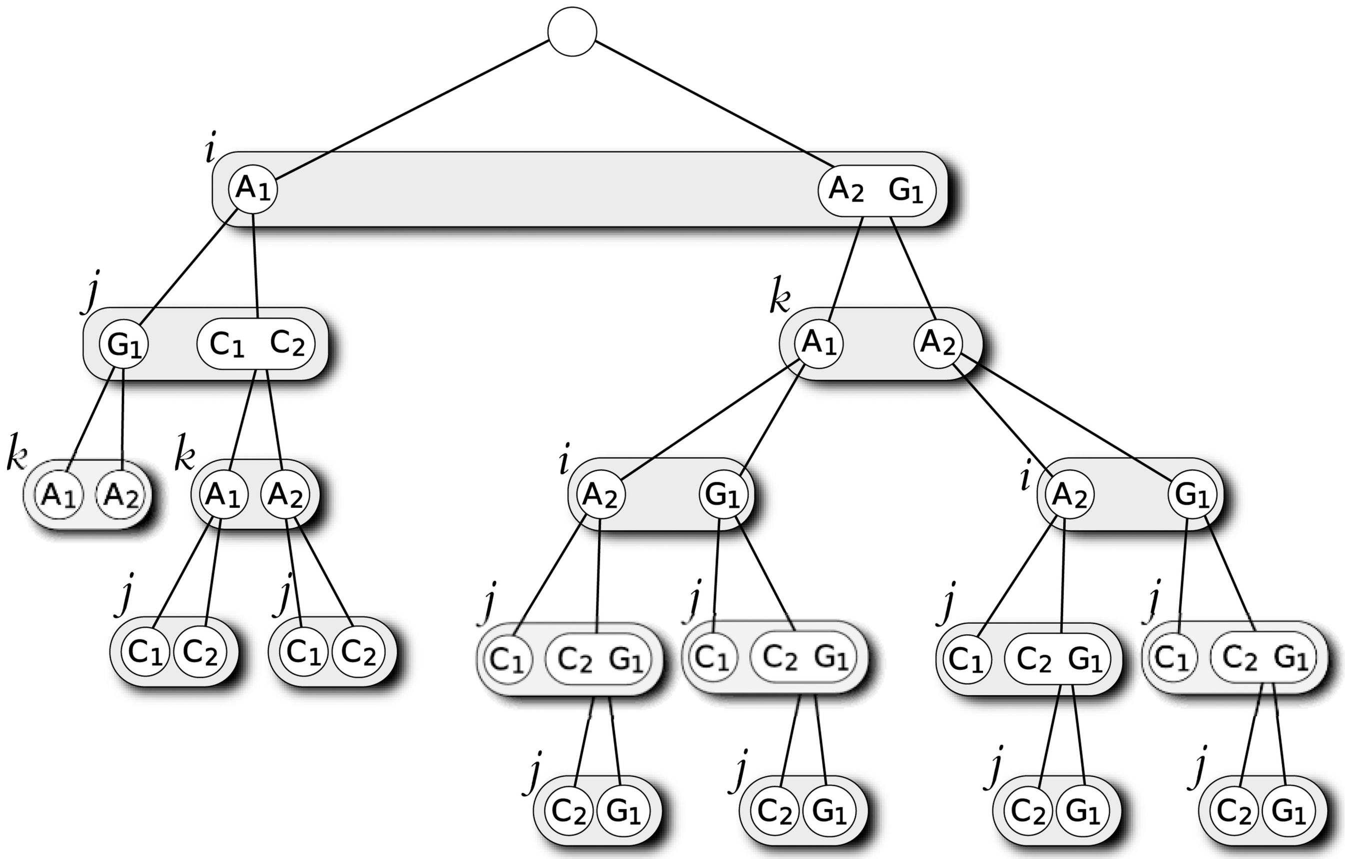

Figure 4.

Binary branching scheme applied to the same naïve CPD problem as in Figure 1. To properly define the tree shape, ordering heuristics must be defined.

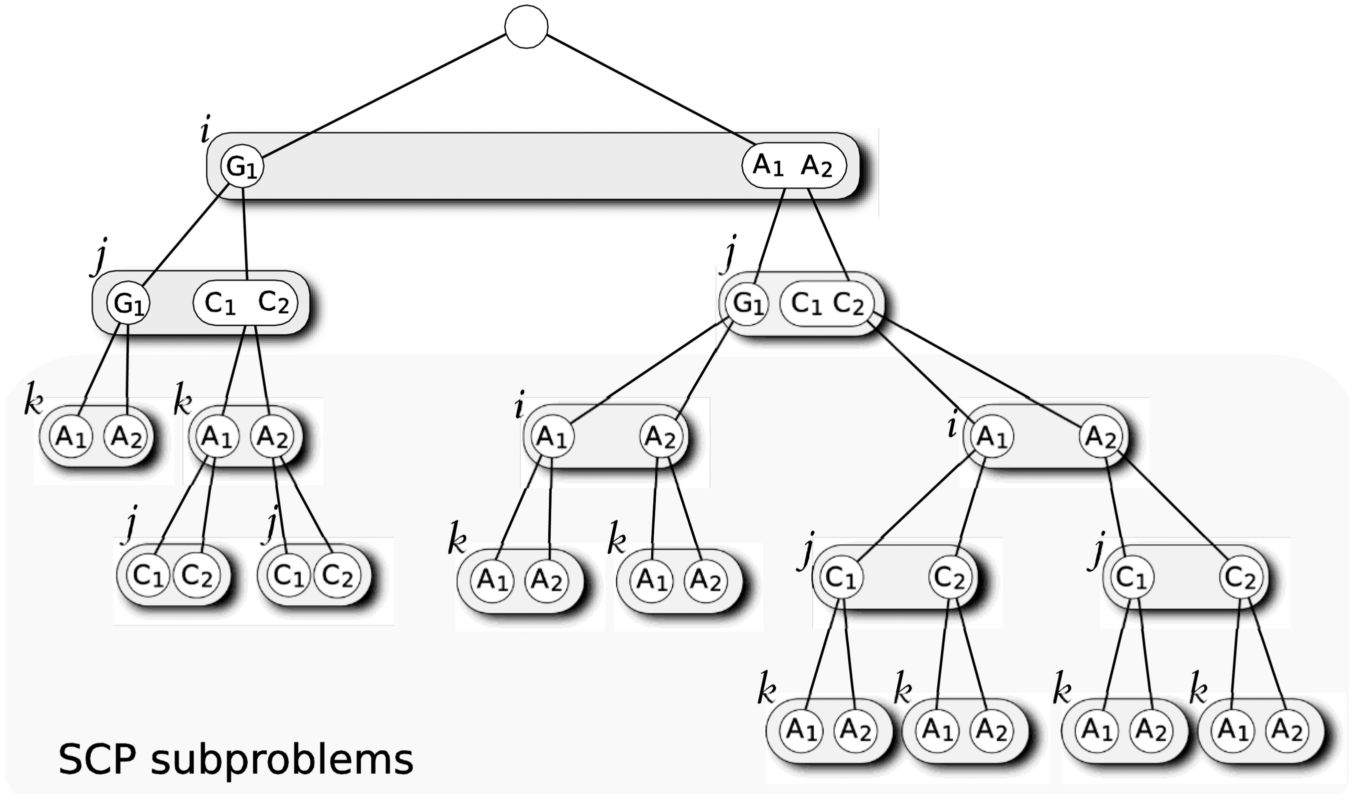

Figure 5.
SCP-branching scheme illustrated on our toy example. Rotamer labels indicate the nature of the amino-acid and the number of the rotamer. Designable position with more than one amino-acid (***i***, ***j***) are selected first. For each such choice, two nodes are added to the search tree: the node on the left restricts available rotamers to a preferred amino acid and the node on the right corresponds to the remaining rotamers. Once all designable positions have their amino-acid type set, search reduces to a SCP problem.

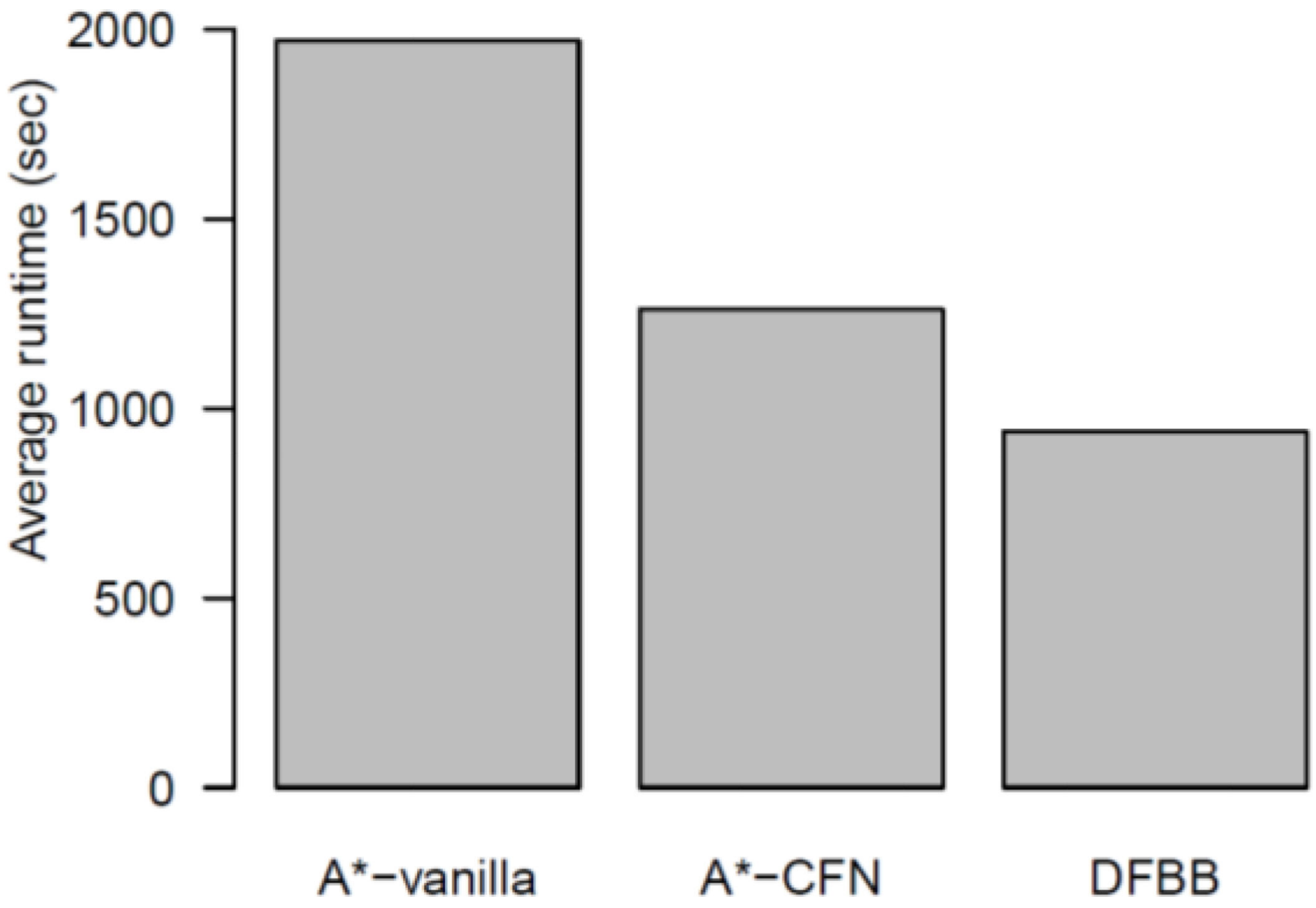

Figure 6.
Comparison of search strategies for solving the GMEC identification problem: the average runtime (in second) for respectively A^*-vanilla; A^*-CFN and DFBB. Cases unsolved by any method are excluded from the comparison. The average runtime is 1970.13; 1262.12 and 938.70 sec for respectively A*-vanilla, A*-CFN and DFBB.

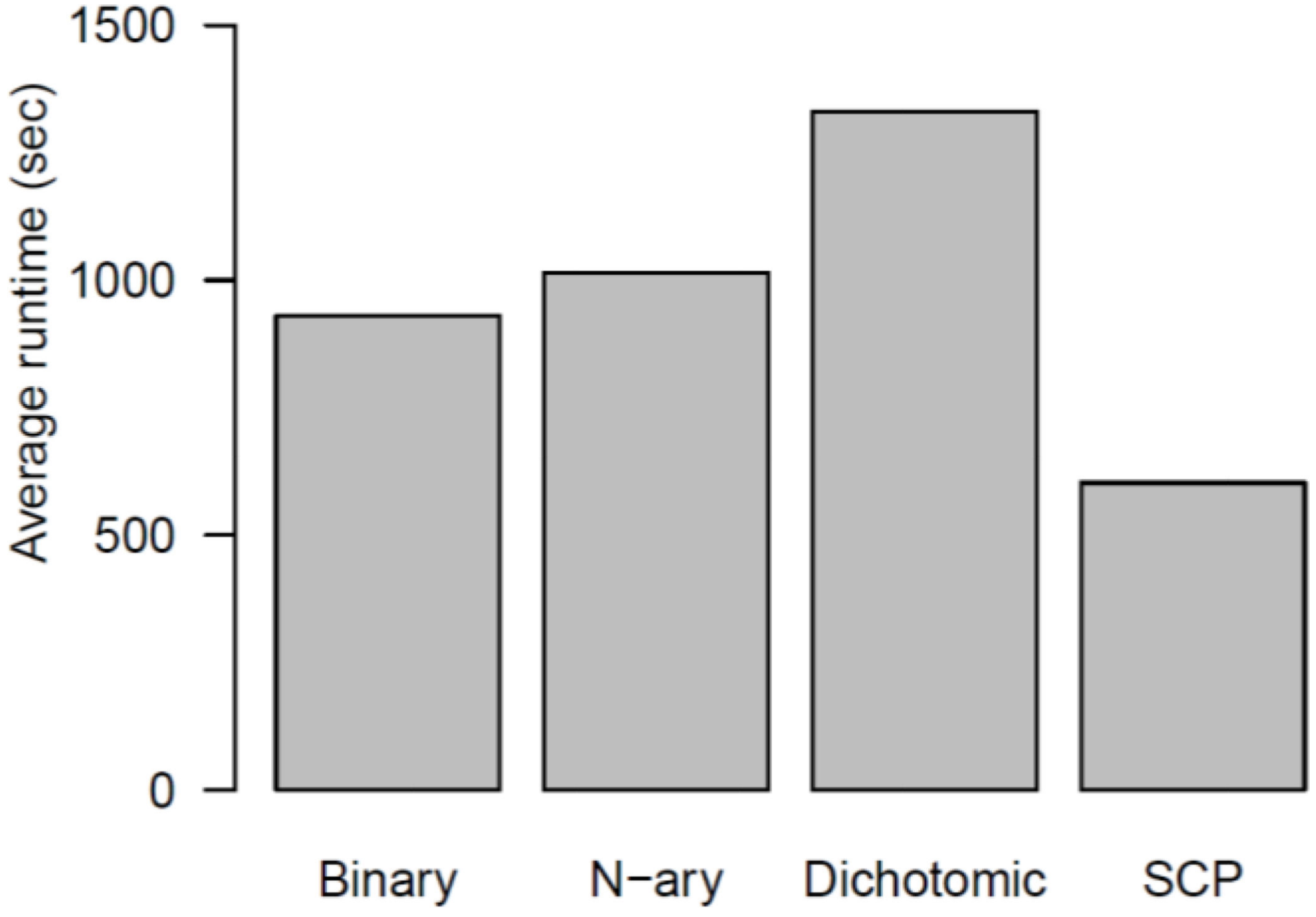

Figure 7.
Comparison of branching schemes for solving the GMEC identification problem. The average runtime is 930.83, 1015.18, 1331.46 and 602.58 sec for respectively Binary, N-ary, Dichotomic and SCP branching schemes.

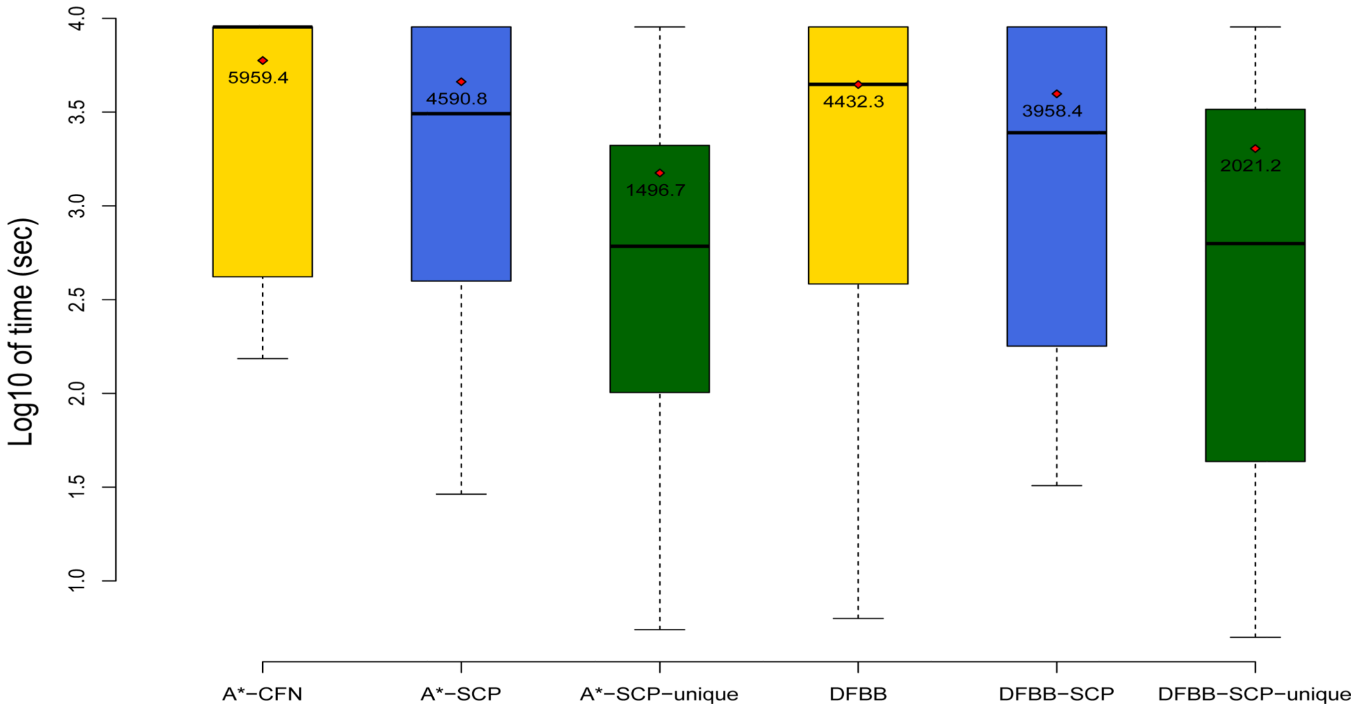

Figure 8.

Boxplot of the runtime (in second) for respectively A^*-CFN; A^*-SCP; A^*-SCP-unique; DFBB; DFBB-SCP and DFBB-SCP-unique. The bold horizontal lines indicate the median (as usual for boxplots). The points in red indicate the average runtimes. The SCP branching schemes significantly improve both A^* and DFBB search strategies. This improvement is even more drastic when the unique sequence constraint is activated.

Table 1

Solving the GMEC identification problem using A^*-vanilla and A^*-CFN. For each method are reported the number of nodes expanded (nd), the runtime in second (time) and the speed (nd/min). Dash line corresponds to cases where no GMEC solution was found in the timeout.

PDB accession code	Sequence - Conformation Space Size	A^* -vanilla			A^*-CFN		
		nd	time (s)	nd/min	nd	time(s)	nd/min
1MJC	4.36E+26	0	5.1	-	0	4.8	-
1CSP	5.02E+30	127	94.6	80	10	38.7	15
1BK2	1.18E+32	5,234	285.3	1,100	21	25.6	49
1SHG	2.13E+32	558	77.5	432	28	76.8	21
1CSK	4.09E+32	113	7.8	869	16	3.7	259
1SHF	1.05E+34	21	13.9	90	8	14.5	33
1FYN	5.04E+36	58,380	3,355.0	1,044	81	160.7	30
1PIN	5.32E+39	24	1,480.4	0	11	1,515.8	0
1NXB	2.61E+41	45	3.8	710	9	2.0	270
1TEN	6.17E+43	243	37.1	392	9	37.9	14
1POH	8.02E+43	169	29.8	340	14	29.1	28
2DRI	1.16E+47	-	-	-	-	-	-
1FNA	3.02E+47	109,523	638.6	10,290	298	571.8	31
1UBI	2.43E+49	193,077	6,895.0	1,680	829	1,913.0	26
1C9O	3.77E+49	63,974	960.6	3,995	20	245.0	4
1CTF	3.95E+51	22,658	2,363.0	575	417	1,949.0	12
2PCY	2.34E+52	14,929	299.1	2,994	119	50.4	141
1DKT	3.94E+58	202,611	1,078.6	11,270	111	855.9	7
2TRX	9.02E+59	29,845	123.5	14,499	245	63.0	233
1CM1	3.73E+63	-	-	-	-	-	-
1BRS	1.67E+64	-	-	-	-	-	-
1CDL	5.68E+65	-	-	-	-	-	-
1LZ1	1.04E+72	-	-	-	4,899	1,481.2	198
1GVP	1.51E+78	-	-	-	-	-	-
1RIS	1.23E+80	-	-	-	-	-	-
2RN2	3.68E+80	15,845	1,276.7	744	104	1,146.3	5

PDB accession code	Sequence - Conformation Space Size	A^* -vanilla			A^*-CFN		
		nd	time (s)	nd/min	nd	time(s)	nd/min
1CSE	8.35E+82	26	79.0	19	8	187.6	2
1HNG	3.70E+88	16	1,739.0	0	4	1,901.5	0
3CHY	2.36E+92	-	-	-	-	-	-
1L63	2.17E+94	19,567	1,409.9	832	74	1,358.7	3
Number of cases solved in 900 sec			14			16	
Number of cases solved in 9,000 sec			22			23	